Introduction to Graphics

Assignment 2: Geometriske Transformationer

Student: Kasper Passov

1 Problem description

1.1 Problem description

We want functions that enable us to transform triangles, so that any triangle can be transformed into any other triangle.

1.2 Scale

Scaling enables us to grow and shrink the triangle, whithout modifying the sides ratio. This is done using matrix transformations with the vector equation
$\begin{bmatrix} xnew \\ ynew \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} xorg \\ yorg \end{bmatrix}$
This is just multiplying the scale factor with the position.

1.3 Rotate

Rotation is found by the vector equation
$\begin{bmatrix} xnew \\ ynew \end{bmatrix} = \begin{bmatrix} cos_\Theta & -sin_\Theta \\ sin\Theta & cos_\Theta \end{bmatrix} \begin{bmatrix} xorg \\ yorg \end{bmatrix}$

1.4 Translocation

1.4.1 Homogen coordinates

The problem with the simple implementation of translocation is it uses addition, where the other transformations uses multiplication. This mean we would have to create speciel cases to handle translocation, which is not an attractive attribute. For this we use Homogeneous Coordinates which is has one ekstra dimension.

1.4.2 Translocate transformation

Translocation under Homogen coordinates is done by
$\begin{bmatrix} wxnew \\ wynew \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ sin\Theta & cos_\Theta & \end{bmatrix} \begin{bmatrix} xorg \\ yorg \end{bmatrix}$

2 Implementation

I used the zipped skeleton *skeletet til geometriske transformationer*, that i made work with a magic makefile i found on the forum.
In the skeleton i created a function that observes keystrokes, and runs glm transformations based on the keystroke. This means i am using the skeletons glm::(scale, rotate and translate) functions instead of implementing my own.

2.1 Comments on implementation

I used the glm::(transformations) to do the transformations needed. Furthermore i created an observer that checks for keystrokes. On each of these keystrokes it does one of the following actions:

- a: Scales the triangle up
- q: Scales the triangle down
- s: Rotates the triangle clockwise
- w: Rotates the triangle counterclockwise
- j: Translocate the triangle left
- l: Translocate the triangle right
- i: Translocate the triangle up
- k: Translocate the triangle down

using the fitting glm::(transformation).

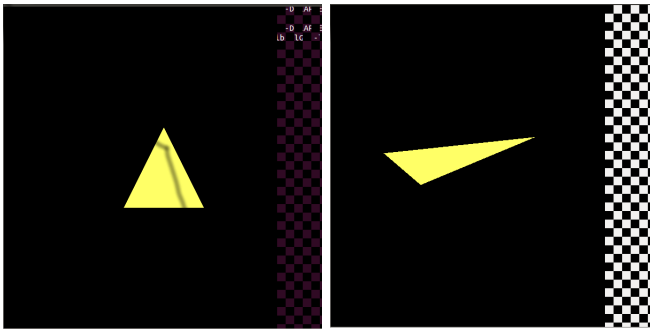
2.2 The implementation

```
static void processNormalKey(unsigned char key, int x, int y) {
    glClear(GL_COLOR_BUFFER_BIT);

        triangleShaderProgram.useProgram();

        glUniformMatrix4fv(glGetUniformLocation(triangleShaderProgram.getProgram(), "uModelM

    modelMatrix = callWithKey(key);

        triangle.draw();
    glutSwapBuffers();
}

static void renderSceneCB() {
    glClear(GL_COLOR_BUFFER_BIT);

        triangleShaderProgram.useProgram();

        glUniformMatrix4fv(glGetUniformLocation(triangleShaderProgram.getProgram(), "uModelM

        glUniform1f(glGetUniformLocation(triangleShaderProgram.getProgram(), "uScale"), 0.25
        glUniform3f(glGetUniformLocation(triangleShaderProgram.getProgram(), "uColour"), 1.0
        triangle.draw();
```

```
    glutSwapBuffers();
}

glm::mat4 callWithKey(unsigned char key) {
    switch(key) {
        case 65: case 97: //a scale down
            return glm::scale(modelMatrix, glm::vec3(0.9f,0.9f,0.0f));
            break;
        case 83: case 115: //s roate neg
            return glm::rotate(modelMatrix, -5.0f, glm::vec3(0.2f,0.2f,0.2f));
            break;
        case 81: case 113: //q scale up
            return glm::scale(modelMatrix, glm::vec3(1.1f,1.1f,0.0f));
            break;
        case 87: case 119: //w roate pos
            return glm::rotate(modelMatrix,  5.0f, glm::vec3(0.2f,0.2f,0.2f));
            break;
        case 106:  //left j
            return glm::translate(modelMatrix, glm::vec3(-0.01f,0.0f,0.0f));
            break;
        case 105:  //up i
            return glm::translate(modelMatrix, glm::vec3(0.0f,0.01f,0.0f));
            break;
        case 108:  //right l
            return glm::translate(modelMatrix, glm::vec3(0.01f,0.0f,0.0f));
            break;
        case 107:  //down k
            return glm::translate(modelMatrix, glm::vec3(0.0f,-0.01f,0.0f));
            break;
        case 27 : //esc
            exit(0);
    }
    return modelMatrix;
```

3 Testing

I used the given framework implementing the keys describer above. Figure 1 (left), is the triangle before any transformations, Figure 2 (right) is after some scaletion, translocation and rotation.



The code has been included in a zipfile and can be run and tried with the commands defined in section 3.1. The program can be exited via the *esc* key